

## Lab 6: Design of a Digital Customer Counter using GPIO Interrupt and LCD Display

### Safety:


In this lab, voltages used are less than 15 volts and this is not normally dangerous to humans. However, you should assemble or modify a circuit when power is disconnected and do not touch a live circuit if you have a cut or break in the skin.

### Learning outcome:

This lab provides an opportunity to learn (a) multiple interrupt handling in an ARM CortexM4F microcontroller and (b) proper configuration to use GPIO external interrupt. The lab uses a TFT LCD display. At the end of the lab, the students will be able to design an interrupt based digital display system for a shop that displays the number of customers present at a given time.

### Required hardware:

Materials	Equipment
None	Stellaris LM4F120 LaunchPad
	1.8" TFT LCD Display (ST7735R)

**Checkpoints:** Throughout this manual, wherever you find a  sign, it means that there is something that you need to (a) show to your instructor and/or (b) observe, measure, and explain in your lab notebook.

### Lab description:

In this lab, you will design a digital display and warning system of a store. The system displays the total customer capacity of the store, which is 30, and the total number of customer inside the store at a given time on the TFT LCD display [4]. If the number of customer is 30 or more, the system will change the display color to red and flash continuously. Assume that there two doors in the store – one for entry, one for exit. We will use two onboard switches, SW1 and SW2, to simulate the entry and exit actions by customers. For example, pressing SW1 once means that a customer has just entered into the store, while pressing SW2 indicates that a customer has just left the store. These switches are active low. You will use GPIO interrupt and handle more than one interrupt requests.

### Interrupt control in Cortex-M:

Interrupts in the LM4F120H5QR MCU are controlled by the Nested Vectored Interrupt Controller (NVIC) (p.47 of [2]). The NVIC and Cortex-M4F prioritize and handle all exceptions in Handler Mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The interrupt vector is fetched in parallel to the state saving (also known as context saving), enabling

efficient interrupt entry. Software can set **eight priority levels** (0 to 7; a higher level corresponds to a lower priority, i.e., level 0 is the highest interrupt priority) on **seven exceptions** (such as, reset, software interrupt, hardware interrupt, bus fault, etc.; more on p.97) and **65 interrupts**. More information on NVIC can be found on p.119 of [2].

The vector addresses of all exceptions can be found on p.98-101 (Table 2-8 and 2-9 in [2]). These definitions are added in the *Startup.s* file provided on the course website [1]. You must download this file and use in your project. Alternatively, you can use the project template provided in Lab 4 or the demo code for GPIO interrupt that already has the required Startup.s file. (\*\*Do not make any changes to the Startup.s file\*\*)

To activate an interrupt source, we need to perform two tasks: (a) enable the source from the corresponding NVIC enable register and (b) set its priority using a priority register. The register maps are given on p.99 and p.129 (see Table 3-8 in [2]). To better understand the activation procedure, an example is given below: Say, you want to use Port F interrupt. First of all, find the interrupt number (i.e., bit position in interrupt register) from Table 2-9 (2nd column) on p.99 corresponding to GPIO Port F. It is “30”. Now find the interrupt register that is needed to enable IRQ30 from Table 3-8 (p.129). It is NVIC\_EN0. So, it means that you need to set bit 30 of NVIC\_EN0 register to enable interrupt on Port F.

From Table 3-8, you will also need to find the register needed to set the priority of IRQ30. It is NVIC\_PRI7. You may choose any priority value from 0 to 7. For both NVIC\_EN0 and NVIC\_PRI7 registers, go to respective pages in [2] to find more instructions on how to use them. You may also review the lecture notes on “interrupt handling” posted on the course website for details [1].

**Interrupt Control for GPIO:** The interrupt capabilities of each GPIO port are controlled by a set of **seven registers**. These registers are used to select the source of the interrupt, its polarity, and edge properties. When one or more GPIO pins cause an interrupt, a single interrupt output is sent to the interrupt controller for the entire GPIO port. For edge-triggered interrupts, software must clear the interrupt to enable any further interrupts. For a level-sensitive interrupt, the external source must hold the level constant for the interrupt to be recognized by the controller. Carefully review section 10.2.2 on p.608 for the complete interrupt handling procedure.

The following three registers are used to define the edge or sense that causes an interrupt:

GPIO\_PORTF\_IS – define interrupt sensing type; 0 = edge, 1 = level (p.617)

GPIO\_PORTF\_IBE – define single edge or both edges (p.618)

GPIO\_PORTF\_IEV – define interrupt event; 0 = falling, 1 = rising (p.619)

Enabling interrupts and checking the status are done using the following four registers:

GPIO\_PORTF\_IM – interrupt mask to individually enable/disable interrupt (p.620)

GPIO\_PORTF\_RIS – raw interrupt status (p.621)

GPIO\_PORTF\_MIS – masked interrupt status (p.622)

GPIO\_PORTF\_ICR – acknowledge/clear interrupt flag by writing ‘1’ (p.623)

**Lab procedure:**

1. The lab requires LCD display driver. Like the previous two labs, you may use the project template of Lab 4 (*Test\_4a\_testingLCD\_webposting.zip*) from the course website. In that case, the settings needed to use the TFT LCD display are already done for you. The main C-code file is “testingLCD.c”.
2. You will now focus on your design. You need the definitions for Port F GPIO registers, NVIC registers, and SYSCCTL\_RCGC2 followed by initialization of these registers. Make sure to enable pull-up resistors on PF0 and PF4, and unlock PF0 to use SW2.



A demo code for GPIO interrupt (using SW1 on PF4) was provided on the course website. You may review the code to verify the configuration for Port F interrupt. Write the register values in your notebook with datasheet reference.

3. Since, you will be using external interrupts generated by Port F, *GPIOPortF\_Handler()* is your interrupt service routine (ISR). So make sure to add *GPIOPortF\_Handler()* function in your main code.

**Handling multiple interrupts:**

In this lab, you will handle multiple interrupts. Once Port F interrupt is activated, pressing any switch will trigger an interrupt and the system will execute the ISR. As a result, you will have to determine the source of interrupt (i.e., whether it is SW1 or SW2) inside the ISR. This can be easily done by checking the 0<sup>th</sup> and 4<sup>th</sup> bits of GPIO\_PORTF\_MIS register. Check these two bits separately and perform necessary actions.

For example, GPIO\_PORTF\_MIS = 0x10 means that the interrupt is triggered by PF4 which indicates that a customer has just entered in the store; so, the current customer count on the display should be incremented. Similarly, GPIO\_PORTF\_MIS = 0x01 tells that PF0 triggered the interrupt and the count should be decremented. Make sure to clear the interrupt flags before exiting the ISR. In this lab, you may not need to use GPIO\_PORTF\_RIS at all.

4. Before you start writing the C-code, it is important to have a big picture of your software. There is no better way to do this than a flowchart. Software flowchart is a type of diagram that shows the tasks performed in a program as boxes. The order of these tasks is shown by connecting the boxes with arrows.



Generate two software flowcharts – one for the main function and one for the ISR. Show the flowcharts to your instructor.

5. Write C-code to accomplish the above mentioned tasks. You may modify “testingLCD.c” if you are using the template. After the C-program is written, compile it and look for errors and warnings. You should use the Stellaris ICDI to debug your code. Ask for assistance if you need help locating the debugging tools. Step through the code and look for errors. Take necessary steps to resolve the issue if any.

6. Obtain from the support office the required hardware for the lab. Connect the LCD display with the LaunchPad. The connection diagram is given in Fig. 1.

The experiment setup is shown in Fig. 2. Note that, during the warning event (i.e., when the number of customer is over the maximum capacity) the display should change the text color to red and flash continuously.

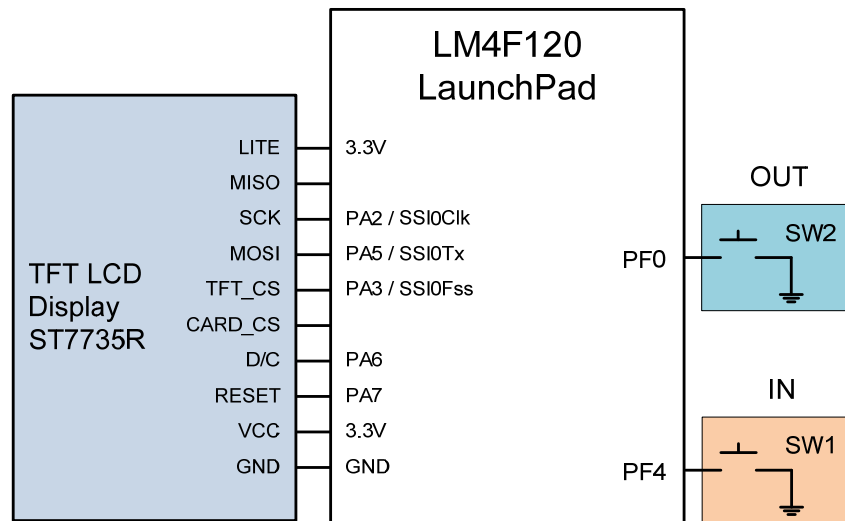


Fig. 1 Connection diagram

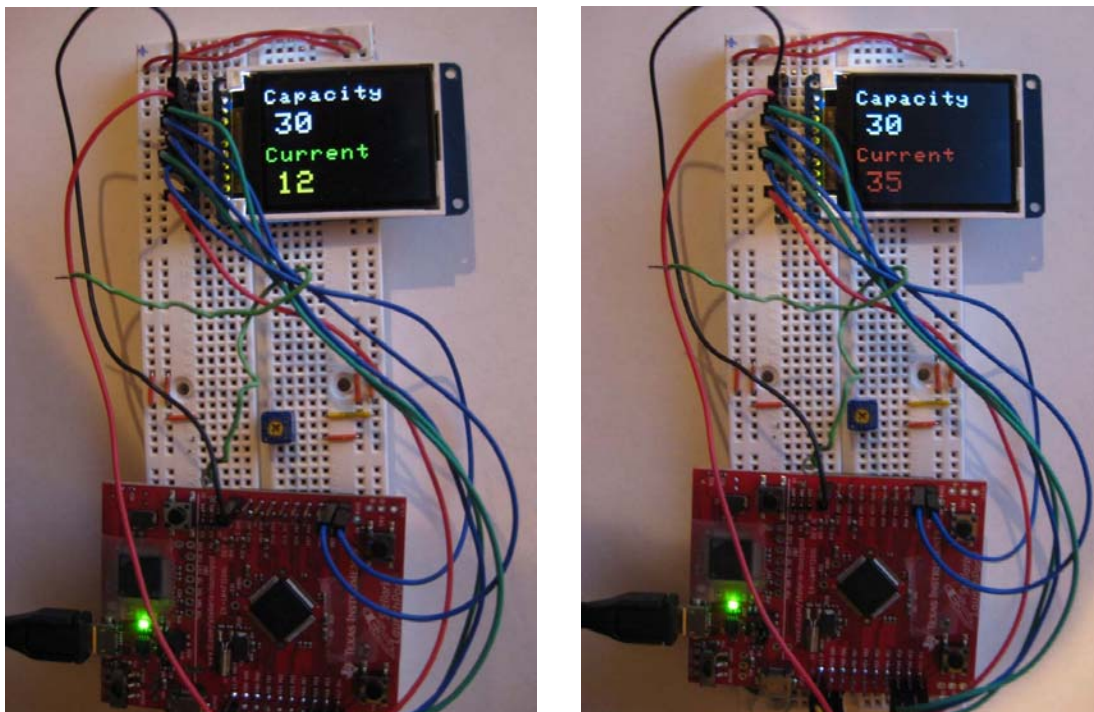





Fig. 2 Experimental setup: left - normal operation; right – flashing warning in red

7.  After all errors are fixed, reload the code into the LaunchPad. Press both switches to simulate customer entry and exit actions. Show your design to the instructor.
8.  In your lab notebook, write down in which part of the lab you used ICDI tools to debug your design. Also say how the ICDI tools helped you resolve the issue.
9.  In your lab notebook, write substantiated conclusions for each portion of the lab. Your conclusion should state what you have learnt in the lab, whether or not the objectives of the learning outcomes are met, and any improvements can be made in your display system.

### References:

1. CME331 class website: <https://www.engr.usask.ca/classes/CME/331>
2. LM4F120H5QR Datasheet (Aug 29, 2012): <http://www.ti.com/lit/gpn/lm4f120h5qr>
3. Stellaris® LM4F120 LaunchPad User Manual (SPMU289A–Revised December 2012): <http://www.ti.com/tool/sw-ek-lm4f120xl>
4. 1.8" 18-bit color TFT LCD display with microSD (ST7735R): <http://www.adafruit.com/products/358>
5. Adafruit GFX Graphics Library: <http://learn.adafruit.com/adafruit-gfx-graphics-library>

### Appendix A: Sensor circuit

The lab uses two switches to simulate the store doors. A sensor circuit could be used to simulate the actual situation. The circuit diagram is given below for your future reference.

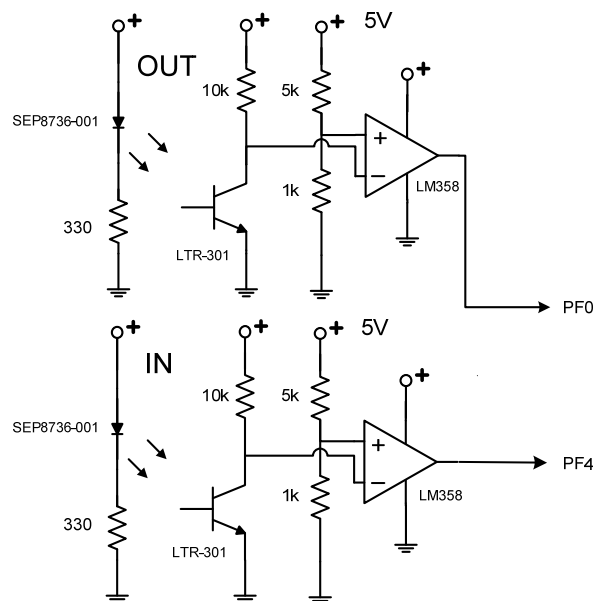


Fig. Sensor circuit for customer counter

## Appendix B: LaunchPad interface

The pin-out diagram of the LaunchPad is given below. Although the LM4F120H5QR MCU has 43 GPIO, only 35 of them are available through the LaunchPad. They are: 6 pins of Port A (PA2-PA7), 8 pins of Port B (PB0-PB7), 4 pins of Port C (PC4-PC7), 6 pins of Port D (PD0-PD3, PD6-PD7), 6 pins of Port E (PE0-PE5), and 5 pins of Port F (PF0-PF4). In addition, there are two ground, one 3.3V, one 5V (VBUS), and one reset pins available on the LaunchPad.

Pins PC0-PC3 are left off as they are used for JTAG debugging. Pins PA0-PA1 are also left off as they are used to create a virtual COM port to connect the LaunchPad to PC. \*\*\*These pins should **not** be used for regular I/O purpose.\*\*\* Besides, in the LaunchPad, PD0 and PD1 are connected with PB6 and PB7 respectively with a  $0\Omega$  resistor (i.e., short circuit, p.20 in [3]). So, user needs to pay special attention when using Ports B and D at the same time.

J1	J3	J4	J2
3.3V	● ●	VBUS	PF2 ● ● GND
PB5	● ●	GND	PF3 ● ● PB2
PB0	● ●	PD0	PB3 ● ● PE0
PB1	● ●	PD1	PC4 ● ● PF0
PE4	● ●	PD2	PC5 ● ● RST
PE5	● ●	PD3	PC6 ● ● PB7
PB4	● ●	PE1	PC7 ● ● PB6
PA5	● ●	PE2	PD6 ● ● PA4
PA6	● ●	PE3	PD7 ● ● PA3
PA7	● ●	PF1	PF4 ● ● PA2

Fig. x Header pins on the LaunchPad (EK-LM4F120XL)

END